

ЛАБОРАТОРНАЯ РАБОТА №1

Обработка исключительных ситуаций в Delphi. Восприятие ввода от пользователя

1. Цель работы

Ознакомление с классом исключительных ситуаций Delphi и создание приложений, генерирующих исключения и обрабатывающие различные фокусы ввода.

2. Домашнее задание

Изучить разделы 5 и 6 конспекта лекций, ознакомиться с описанием и заданием на лабораторную работу.

3. Основные понятия и приемы

3.1. Обработка исключений

На этапе выполнения Delphi порождает исключения, когда какой-либо процесс идет неправильно. Если код вашей подпрограммы написан соответствующим образом, он может распознать возникшую проблему и попытаться ее решить; в противном случае исключение передается в код, который вызвал вашу подпрограмму и т.д. В конечном счете, если никто не обработал исключение, его обрабатывает Delphi, выводя на экран стандартное сообщение об ошибке и пытаясь продолжить выполнение программы.

Весь механизм строится на четырех ключевых словах:

- ✓ try – определяет начало защищенного блока кода;
- ✓ except – определяет конец защищенного блока кода и вводит операторы обработки исключений в следующем виде:

on (тип исключения) do (оператор)

- ✓ finally – указывает необязательный блок, который используется для освобождения ресурсов, распределенных в блоке try перед обработкой исключения; этот блок завершается ключевым словом end.
- ✓ raise – оператор, используемый для порождения исключений. Большинство исключений, которые вы встретите при программировании на Delphi, будут порождаться системой, но вы также можете создать их в собственном коде, когда во время выполнения обнаружатся недопустимые или несогласованные данные. Кроме того, ключевое слово raise можно использовать внутри обработчика для повторного порождения исключе-

ния, т.е. для передачи его следующему обработчику.

Если вы хотите, чтобы при правильной обработке исключений программа продолжала выполняться, отключите опцию отладки Break on Exception в окне Environment Options.

3.2. Предопределенные обработчики исключительных ситуаций

Ниже Вы найдете справочную информацию по предопределенным исключениям, необходимую для профессионального программирования в Delphi.

- **Exception** - базовый класс-предок всех обработчиков исключительных ситуаций.
- **EAbort** - “скрытое” исключение. Используйте его тогда, когда хотите прервать тот или иной процесс с условием, что пользователь программы не должен видеть сообщения об ошибке. Для повышения удобства использования в модуле *SysUtils* предусмотрена процедура *Abort*, определенная как:

```
procedure Abort;  
begin  
    raise EAbort.CreateRes(SOperationAborted) at ReturnAddr;  
end;
```
- **EComponentError** - вызывается в двух ситуациях:
 - 1) при попытке регистрации компоненты за пределами процедуры Register;
 - 2) когда имя компоненты не уникально или не допустимо.
- **EConvertError** - происходит в случае возникновения ошибки при выполнении функций *StrToInt* и *StrToFloat*, когда конвертация строки в соответствующий числовой тип невозможна.
- **EInOutError** - происходит при ошибках ввода/вывода при включенной директиве *{SI+}*.
- **EIntError** - предок исключений, случающихся при выполнении целочисленных операций.
- **EDivByZero** - вызывается в случае деления на ноль, как результат RunTime Error 200.
- **EIntOverflow** - вызывается при попытке выполнения операций, приводящих к переполнению целых переменных, как результат RunTime Error 215 при включенной директиве *{SQ+}*.
- **ERangeError** - вызывается при попытке обращения к элементам массива по индексу, выходящему за пределы массива, как результат RunTime Error 201 при включенной директиве *{SR+}*.
- **EInvalidCast** - происходит при попытке приведения переменных од-

ного класса к другому классу, не совместимому с первым (например, приведение переменной типа TListBox к TMemo).

- **EInvalidGraphic** - вызывается при попытке передачи в *LoadFromFile* файла, несовместимого графического формата.
- **EInvalidGraphicOperation** - вызывается при попытке выполнения операций, неприменимых для данного графического формата (например, Resize для TIcon).
- **EListError** - вызывается при обращении к элементу наследника TList по индексу, выходящему за пределы допустимых значений (например, объект TStringList содержит только 10 строк, а происходит обращение к одиннадцатому).
- **EMathError** - предок исключений, случающихся при выполнении операций с плавающей точкой.
- **EOverflow** - происходит как результат переполнения операций с плавающей точкой при слишком больших величинах. Соответствует RunTime Error 205.
- **Underflow** - происходит как результат переполнения операций с плавающей точкой при слишком малых величинах. Соответствует RunTime Error 206.
- **EZeroDivide** - вызывается в результате деления на ноль.
- **EMenuError** - вызывается в случае любых ошибок при работе с пунктами меню для компонент TMenu, TMenuItem, TPopupMenu и их наследников.
- **EOutlineError** - вызывается в случае любых ошибок при работе с TOutLine и любыми его наследниками.
- **EOutOfMemory** - происходит в случае вызовов New(), GetMem() или конструкторов классов при невозможности распределения памяти. Соответствует RunTime Error 203.
- **EPrinter** - вызывается в случае любых ошибок при работе с принтером.
- **EProcessorException** - предок исключений, вызываемых в случае прерывания процессора- hardware breakpoint. Никогда не включается в DLL, может обрабатываться только в “цельном” приложении.
- **EBreakpoint** - вызывается в случае останова на точке прерывания при отладке в IDE Delphi. Среда Delphi обрабатывает это исключение самостоятельно.
- **EFault** - предок исключений, вызываемых в случае невозможности обработки процессором тех или иных операций.
- **EInvalidOpCode** - вызывается, когда процессор пытается выполнить недопустимые инструкции.
- **ESingleStep** - аналогично EBreakpoint, это исключение происходит

при пошаговом выполнении приложения в IDE Delphi.

- **EPropertyError** - вызывается в случае ошибок в редакторах свойств, встраиваемых в IDE Delphi. Имеет большое значение для написания надежных property editors. Определен в модуле DsgnIntf.pas.
- **EResNotFound** - происходит в случае тех или иных проблем, имеющих место при попытке загрузки ресурсов форм - файлов .DFM в режиме дизайнера. Часто причиной таких исключений бывает нарушение соответствия между определением класса формы и ее описанием на уровне ресурса (например, вследствие изменения порядка следования полей-ссылок на компоненты, вставленные в форму в режиме дизайнера).
- **EStreamError** - предок исключений, вызываемых при работе с потоками.
- **EFCREATEError** - происходит в случае ошибок создания потока (например, при некорректном задании файла потока).
- **EFileError** - вызывается при попытке вторичной регистрации уже зарегистрированного класса (компоненты). Является, также, предком специализированных обработчиков исключений, возникающих при работе с классами компонент.

3.3. Восприятие ввода от пользователя

Обратим особое внимание на одно качество, характерное для многих управляющих элементов – фокус. Как определить какое окно или поле имеет фокус ввода? В каждый конкретный момент фокус имеет только одно поле. Вы можете перемещать фокус, используя клавишу Tab или щелкая мышью по другому компоненту. Каждый раз когда компонент получает или теряет фокус, к нему приходит соответствующее событие, которое указывает, что пользователь достиг (OnEnter) или покинул (OnExit) компонент.

При использовании компонента Edit для ввода чисел пользователь вместо цифры может набрать букву. Функции преобразования вернут код ошибки, что поможет определить действительно ли введено число. Когда можно выполнить такую проверку? Возможно, когда изменится значение блока редактирования, когда компонент потеряет фокус или когда пользователь щелкнет по некоторой кнопке в диалоговой панели. Можно просматривать входной поток в блоке редактирования и останавливать любой нечисловой код.

4. Порядок выполнения работы

1. В среде программирования Delphi создайте новый проект, выбрав пункт меню File/New Application.

2. Сохраните этот проект в папке "C:\Ваша_группа\Ваша_фамилия Lab2". (Unit1.pas под новым именем Main2.pas, а Project1.dpr под новым именем Lab2.dpr).
3. Разработайте приложение, обрабатывающее исключительную ситуацию, согласно вашему варианту индивидуального задания.
4. Открыть новое приложение.
5. Создать форму с пятью полями редактирования и пятью соответствующими надписями, которые поясняют, какой вид проверки осуществляет соответствующий компонент Edit. Форма также содержит кнопку для проверки содержимого первого поля редактирования.
Событие OnClick кнопки должно проверять целочисленность введенного в первое поле значения, например:

```

var
  Number, Code : Integer ;
begin
  if Edit1.Text <> '' then
    begin
      val ( Edit1. Text, Number, Code ) ;
      if Code <> 0 then
        begin
          Edit1. SetFocus ;
          MessageDlg ( ' Not a number in the first edit ' , mtError,
            [ mbOK ] , 0 ) ;
        end ;
      end ;
    end ;
end ;

```

- ◆ При выходе из компонента Edit2 (событие OnExit) осуществляется аналогичная проверка.

```

var
  Number, Code : Integer ;
begin
  if (Sender as TEdit ). Text <> '' then
    begin
      val ((Sender as TEdit ). Text, Number, Code) ;
      if Code <> 0 then
        begin
          (Sender as TEdit ). SetFocus ;
          MessageDlg ( ' The edit field number ' +
            IntToStr ((Sender as TEdit ). Tag) +
            ' does not have a valid number' , mtError, [ mbOK ] , 0 ) ;
        end ;
      end ;
    end ;
end ;

```

end ;

Текст сообщения об ошибке может варьироваться.

- ◆ Третий компонент Edit выполняет аналогичную проверку при каждом изменении его содержимого (используя событие OnChange).
- ◆ Записать код для события события OnKeyPress компонента Edit4 и проверить, не является ли введенный символ числом или клавишей Backspace (которая имеет числовое значение 8).

```
begin
  if not ( key in [ '0' . . '9' , # 8 ] ) then
    begin
      Key := # 0 ;
      MessageBeep ($ FFFFFFFF) ;
    end;
  end;
end;
```

- ◆ Для события OnEnter компонента Edit5 записать код, в котором необходимо преобразовать введенные символы в число с помощью функции StrToInt. Использовать исключение для обработки ошибки EConvertError.

5. Варианты индивидуальных заданий

1. Создать программу, позволяющую пользователю ввести два числа, которые программа разделит. Необходимо поместить на форму три объекта класса TEdit - два для операндов, один – для результата и кнопку (объект класса TButton), нажимая на которую пользователь выполняет деление. Исключить попытку деления на ноль а так же введения символов вместо цифр. Выдать сообщение о типе возникшей ошибке.
2. Создать программу, вычисляющую корни квадратного уравнения ($ax^2+bx+c=0$). Необходимо поместить на форму четыре объекта класса TEdit - три для коэффициентов квадратного уравнения, один – для результата и кнопку (объект класса TButton), нажимая на которую пользователь выполняет нахождение корней. Исключить ввод символов вместо цифр, получение отрицательного дискриминанта и ввод $a = 0$. Вывести при всех типах ошибок одно и то же сообщение.
3. Создать программу с “бесконечным” циклом типа while. В цикле увеличивать переменную I до значения, заданного пользователем. При достижении этого значения выходить из цикла с помощью возбуждения исключения EAbort. Выдать сообщение о выходе из цикла в блоке Except. Необходимо поместить на форму кнопку (объект класса TButton), которая запускает цикл; сообщение можно выдать с

помощью функции ShowMessage, или поместить на форму метку (объект класса TLabel), в которую помещается сообщение.

4. Создать программу, вычисляющую тангенс угла. Необходимо поместить в форму два компонента Tedit для ввода значения и результата и кнопку Tbutton для вычисления значения тангенса. Исключить ввод символов вместо цифр и получение значения тангенса угла 90 градусов. Предусмотреть возможность ввода значений в радианах.
5. Создать программу, вычисляющую логарифм числа. Для этого необходимо поместить в форму два компонента Tedit для ввода значения и результата и кнопку Tbutton для вычисления значения логарифма. Исключить ввод символов вместо цифр и получение значения логарифма 0.
6. Создать программу обработки исключения при обращении к несуществующему элементу массива. В форму поместите поля редактирования для ввода – вывода значений и номеров элементов массива и кнопку для обработки события.

6. Результаты работы

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, содержащий обработку исключительной ситуации, файл формы и исходный код модуля.

ЛАБОРАТОРНАЯ РАБОТА №2

Создание и обработка меню

1. Цель работы

Ознакомление с дизайнером меню Delphi и создание приложения, содержащего меню.

2. Домашнее задание

Изучить 7 раздел конспекта лекций, ознакомиться с описанием и заданием на лабораторную работу.

3. Основные понятия и приемы

3.1. Структура меню

Обычно меню имеет два уровня. Строка меню, которая находится под заголовком окна, содержит имена выпадающих меню. Каждое выпадающее меню содержит несколько элементов. Однако структура меню очень гибка. Элемент меню можно поместить непосредственно в строку меню, а выпадающее меню внутри другого выпадающего меню. Выпадающее меню внутри другого выпадающего меню (выпадающее меню второго уровня) встречается довольно часто, и для этого случая Windows предоставляет специальный визуальный значок – маленький треугольник справа от соответствующего пункта меню. Нередко вместо выпадающего меню второго уровня вы можете просто сгруппировать ряд опций в первоначальном выпадающем меню и поместить два разделителя: один до группы и один после.

3.2. Различные роли элементов меню

Существует три основных типа элементов меню:

- ◆ Команды – элементы меню, которые используются для выдачи команды и выполнения действия. Визуально они никак не выделяются.
- ◆ Установщики состояния – элементы меню, которые используются для переключения опции в положения включено – выключено и изменения состояния какого – либо элемента. Если эти команды имеют два состояния, то в активном положении слева от них обычно стоит галочка. В этом случае выбор команды изменяет состояние на противоположное.
- ◆ Элементы вызова диалога – элементы меню, которые вызывают диалоговую панель. Реальное различие между этими и другими элементами меню состоит в следующем: с помощью этих элементов пользователь должен получить возможность исследовать вероятные действия соответствующей диалоговой панели. Такие команды должны иметь визуальный ключ в виде трех точек после текста.

3.3. Редактирование меню с помощью Menu Designer

Система Delphi включает специальный редактор для меню Menu Designer. Чтобы вызвать этот инструмент, поместите компонент меню в форму и дважды щелкните по нему. Не волнуйтесь о точном положении данного компонента в форме, поскольку на результат это не влияет: само меню всегда помещается правильно – под заголовком формы. Menu Designer позволяет создавать меню путем простого написания текста команд, перемещать элементы или выпадающие меню с помощью буксировки и легко устанавливать свойства элементов. Для создания выпадающего меню второго уровня нужно выбрать команду Create submenu в SpeedMenu инструмента (локальном меню, которое вызывается правой кнопкой мыши).

3.4. Горячие клавиши меню

Общее свойство элементов меню – наличие подчеркнутой буквы. Эту букву можно использовать для выбора меню с помощью клавиатуры. При одновременном нажатии клавиши Alt и клавиши с буквой на экране появляется соответствующее выпадающее меню. Чтобы определить подчеркнутую клавишу, просто поместите перед ней символ амперсанта (&), например, &File. Элементам меню можно назначить горячие клавиши. Для этого нужно указать значение для свойства ShortCut, выбрав одну из стандартных комбинаций.

3.5. Изменение элементов меню

Для модификации элемента меню чаще всего используются три свойства. Свойство Checked используется, чтобы добавить или удалить галочку рядом с элементом меню. С помощью свойства Enabled элемент меню можно пригасить, после чего пользователь не сможет его выбрать. Свойство Caption представляет текст элемента меню. Изменяя текст элемента меню, вы указываете пользователю, что программа перешла в другое состояние.

4. Порядок выполнения работы

1. Войдите в среду Delphi.
2. Выберите пункт меню File/New Application, этим вы создадите новый проект приложения.
3. Выберите команду File/Save Project As...
 - ◆ В появившемся диалоге перейдите к корневому каталогу диска C, выбрав соответствующий пункт в выпадающем списке вверху окна.
 - ◆ Откройте папку, соответствующую названию вашей группы (например, Р-123). Если такой нет, то создайте её, кликнув правой кнопкой мыши на свободном месте, выбрав из появившегося контекстного меню пункт Создать/папку и введя нужное название (после чего не

забудьте её открыть).

- ♦ Создайте (как описано в предыдущем пункте) папку "Ваша_фамилия Lab3".
- ♦ Сохраните Unit1.pas под новым именем Main.pas, а Project1.dpr под новым именем Lab3.dpr.
- ♦ Создайте проект согласно Вашему индивидуальному заданию. Подробное описание разработки меню приведено в индивидуальном задании 1.

5. Варианты индивидуальных заданий

Вариант 1

1. Выберите в палитре компонентов Delphi вкладку Standard (это самая первая вкладка и обычно она уже выбрана автоматически).
2. Выберите компонент MainMenu и поместите его на форму.

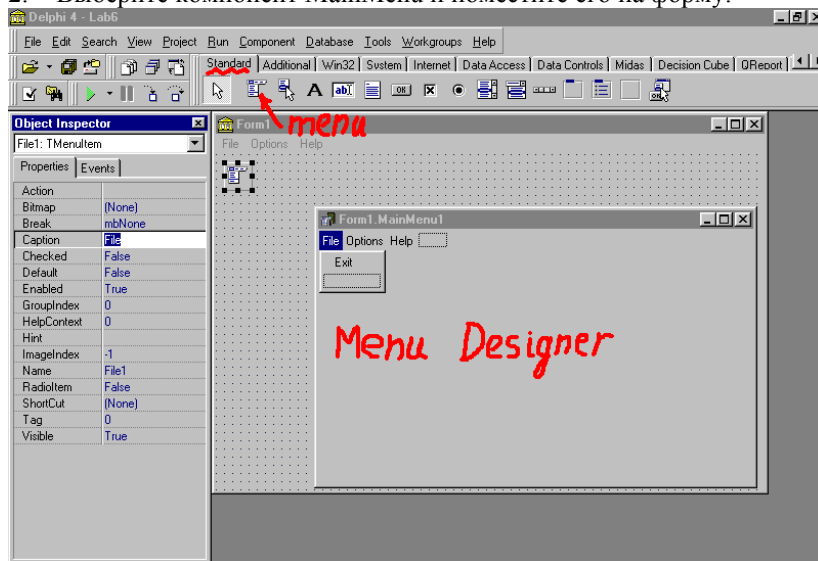


Рис. 3.1. Дизайнер меню

3. Откройте Menu Designer дважды кликнув на компоненте MainMenu в форме. Окно дизайнера меню представлено на рис. 3.1.
4. Создайте меню следующей структуры:
 - ♦ File: Exit
 - ♦ Options: Font, Color, (разделитель), Left, Center, Right, (разделитель), Height
 - ♦ Help: About

Меню должно быть похоже на представленное на рис. 3.2.
 Названия пунктов меню вводятся в поле Caption, для создания разделителя в поле Caption вводится символ "-" (минус, он же дефис)



Рис. 3.2. Пример меню

5. Для Left, Center и Height назначьте горячие клавиши (свойство ShortCut)
6. В форму поместите компонент RichEdit из палитры Win32 и две пиктограммы диалогов: FontDialog и ColorDialog из палитры Dialogs.
7. Для отклика на команды меню вы должны определить метод для события OnClick каждого элемента меню, как показано на рис. 3.3.

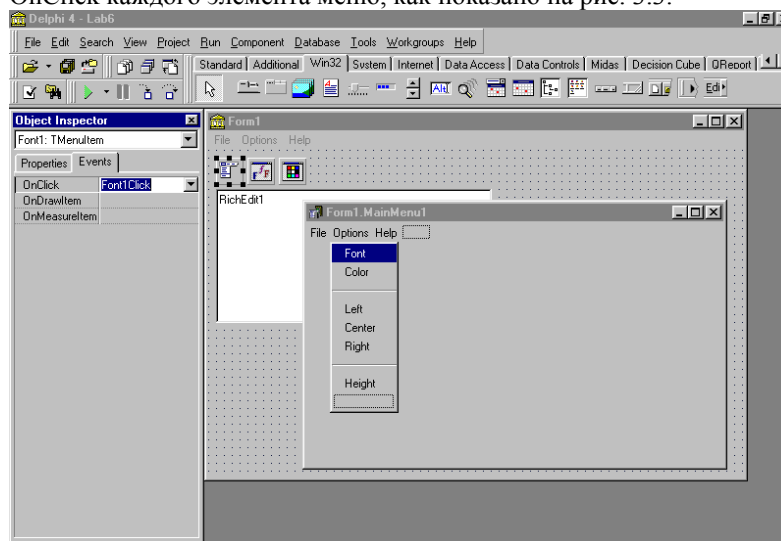


Рис. 3.3. Выбор метода OnClick

8. Код метода TForm1.Font1Click:

```
procedure TForm1.Font1Click (Sender : TObject) ;
begin
    FontDialog1.Font := RichEdit1.Font ;
```

```

FontDialog1.Execute;
RichEdit1.Font := FontDialog1.Font ;
end;

```

9. Код метода TForm1.Color1Click очень похож на приведённый выше. Напишите его сами.
10. Код метода TForm1.Left1Click:

```

procedure TForm1.Left1Click (Sender : TObject) ;
begin
    RichEdit1.Paragraph.Alignment := taLeftJustify ;
    Left1.Checked := True ;
    Center1.Checked := False ;
    Right1.Checked := False ;
end;

```

Чтобы поставить галочку в ряде выбираемых опций, установите свойство Checked элемента меню в True, как показано на рис. 3.4.

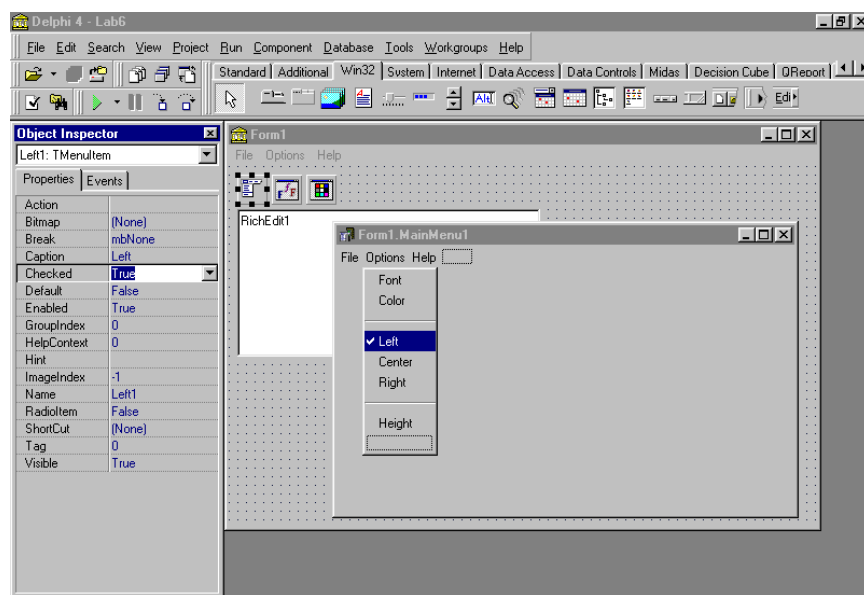


Рис. 3.4. Параметры инспектора объектов

11. Коды методов для элементов Center и Right аналогичны предыдущему.

Вариант 2

1. На форме расположите две панели , две кнопки и компонент RichEdit. Первая панель должна содержать два поля редактирования, а вторая два чекбокса, как показано на рис. 3.5.

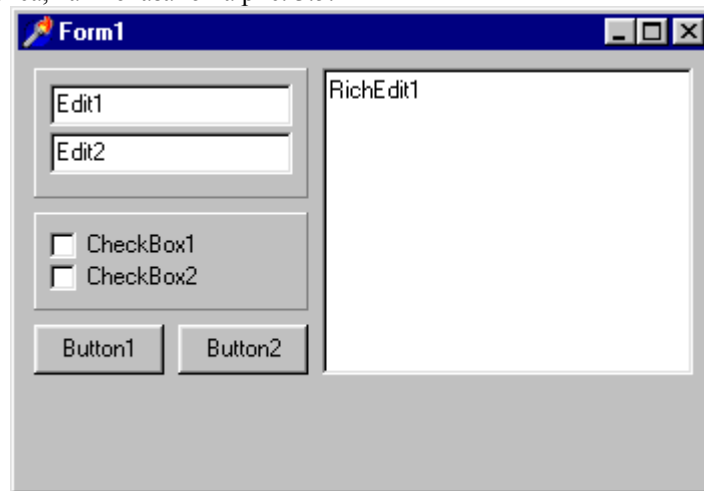


Рис. 3.5. Начальная форма проекта

2. Создайте меню:
 - ◆ File: Open, Save As
 - ◆ Buttons: Enable First (программно изменяемый на Disable First и обратно; как именно – см. ниже)
 - ◆ Views: Edit Boxes, Check Boxes
 - ◆ Pulldowns: Remove File Menu, Disable Buttons Menu
3. Поместите в форму пиктограммы диалогов OpenFileDialog, SaveDialog. Форма будет выглядеть примерно так, как показано на рис.3.6.

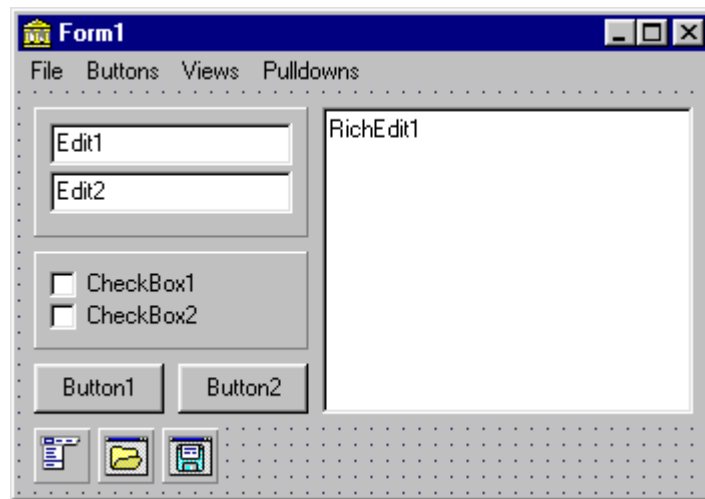


Рис. 3.6. Законченный вид формы проекта

4. Код методов, которые загружают и сохраняют файлы :

```
procedure TForm1.Open1Click (Sender : TObject) ;
begin
  if OpenFileDialog1.Execute then
    RichEdit1.Lines.LoadFromFile (OpenDialog1.FileName) ;
end;

procedure TForm1.SaveAs1Click (Sender : TObject) ;
begin
  if SaveDialog1.Execute then
    RichEdit1.Lines.SaveToFile (SaveDialog1.FileName) ;
end;
```

5. Компоненты внутри панелей в действительности не используются. Однако вам необходимо воспользоваться двумя кнопками, чтобы скрыть или отобразить каждую из двух панелей вместе с управляющими элементами, которые в них содержатся. Те же действия можно выполнить с помощью двух команд меню : View/ Edit Boxes и View/ Check Boxes. Когда вы выбираете одну из этих команд меню или нажимаете одну из кнопок, происходят три разных действия. Во-первых, отображается или скрывается панель. Во-вторых, текст кнопки изменяется с Hide на Show, и наоборот. В-третьих, рядом с соответствующим элементом меню появляе-

ся или исчезает галочка. Ниже приведен код одного из двух методов, который связан с событиями щелчка как команды меню, так и кнопки :

```
procedure TForm1.ViewEdit1Click (Sender : TObject) ;
begin
  Panel1.Visible := not Panel1.Visible ;
  ViewEdit1.Checked := not ViewEdit1.Checked ;
  if Panel1.Visible then
    Button1.Caption := 'Hide' ;
  else
    Button1.Caption := 'Show' ;
end;
```

6. Команды меню Buttons применяют другой подход. Для показа текущего состояния они используют не галочку, а изменение текста. Кроме того, они разрешают или запрещают соответствующую команду View и кнопку.

```
procedure TForm1.ButtonsFirst1Click (Sender : TObject) ;
begin
  if Buttons1.Enabled then
    begin
      Buttons1.Enabled := False;
      ViewEdit1.Enabled := False ;
      ButtonsFirst1.Caption := 'Enable &First' ;
    end
  else
    begin
      Buttons1.Enabled := True
      ViewEdit1.Enabled := True ;
      ButtonsFirst1.Caption := 'Disable &First' ;
    end
end;
```

7. Команды меню Pulldowns должны скрывать выпадающее меню File и Buttons соответственно и показывать галочку для выбранного элемента. Запишите код для каждого элемента этого меню самостоятельно.

Вариант 3

1. На форме расположите компонент Image, пиктограмму диалога OpenPictureDialog

2. Создайте меню (рис. 3.7):
- ◆ File: Open, (разделитель), Exit, (разделитель) - невидимый, Most Recent - невидимый
 - ◆ Options: Center, Stretch, Transparent
 - ◆ About

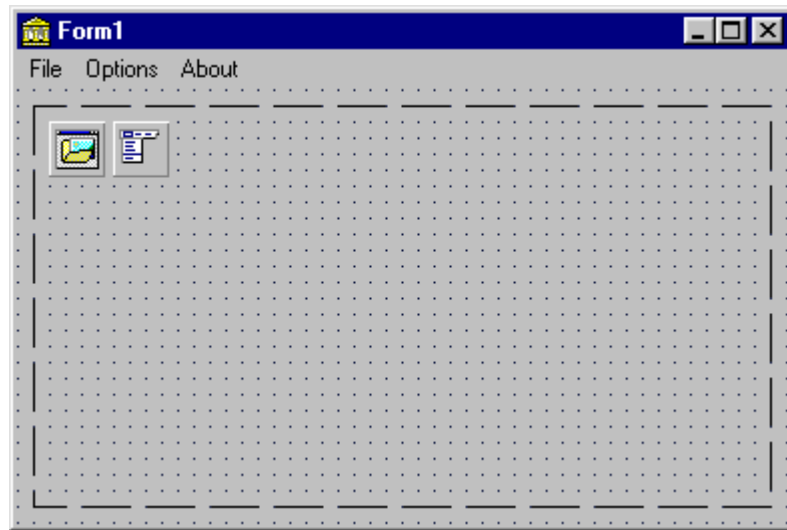


Рис. 3.7. Форма проекта варианта 3

3. Методы меню File:

```
procedure TForm1.Open1Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
  begin
    if Current<>" then
    begin
      MostRecent1.Caption := Current;
      N2.Visible := true;
      MostRecent1.Visible := true;
    end;
    Current := OpenPictureDialog1.FileName;
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
  end;
end;
```



```

procedure TForm1.MostRecent1Click(Sender: TObject);
var
  S : string;
begin
  S := MostRecent1.Caption;
  Image1.Picture.LoadFromFile(S);
  MostRecent1.Caption := Current;
  Current := S;
end;

```

4. Метод меню TForm1.Center1Click:

```

procedure TForm1.Center1Click(Sender: TObject);
begin
  Center1.Checked := not Center1.Checked;
  Image1.Center := Center1.Checked;
end;

```

5. Остальные два метода меню Options аналогичны приведённому выше TForm1.Center1Click
6. Измените меню File и соответствующие методы для отображения имён и повторного открытия не одной, а трёх последних картинок

Вариант 4

1. На форме расположите компоненты Edit и Button
 2. Поместите в форму компонент RadioGroup с тремя пунктами: Beep, About, Exit
 6. Создайте меню:
 - ◆ Normal Part: Beep, About, Exit
 - ◆ Anomal Part
- Пример формы представлен на рис. 3.8.

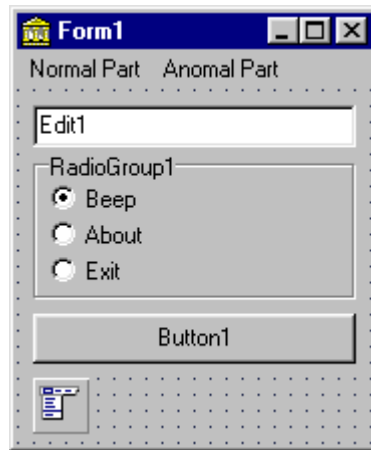


Рис. 3.8. Форма проекта варианта 4

4. Коды методов TForm1.Beep1Click и TForm1.Button1Click:

```

procedure TForm1.Beep1Click(Sender: TObject);
begin
  MessageBeep($FFFFFFFF);
  if (Sender as TMenuItem).Owner=AnomalPart1 then Anomal-
Part1.Remove(Sender as TMenuItem);
end;

procedure TForm1.About1Click(Sender: TObject);
begin
  Application.MessageBox('!!!', '...', 0);
  if (Sender as TMenuItem).Owner=AnomalPart1 then Anomal-
Part1.Remove(Sender as TMenuItem);
end;

procedure TForm1.Exit1Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  I : TMenuItem;
begin
  I := TMenuItem.Create(AnomalPart1);

```

```

I.Caption := Edit1.Text;
case RadioGroup1.ItemIndex of
  0: I.OnClick := Beep1Click;
  1: I.OnClick := About1Click;
  2: I.OnClick := Exit1Click;
end;
AnomalPart1.Add(I);
end;

```

5. Коды методов TForm1.About1Click аналогичен коду метода TForm1.Beep1Click
6. Добавьте в форму компонент ImageList и добавьте в него несколько иконок или небольших картинок
7. Установите свойство меню Images указывающим на ImageList1
8. Задайте свойство ImageIndex для некоторых пунктов меню.
9. Сделайте так, чтобы любой пункт из аномальной части меню случайным образом менял свойства Caption, Checked или ImageIndex какого-либо из оставшихся элементов, если таковые имеются

Вариант 5

1. На форме расположите компоненты Memo и Image, пиктограммы диалогов OpenFileDialog, SaveDialog и OpenPictureDialog.
2. Создайте меню:
 - ◆ Text: Load, Save, (разделитель), Enabled – с галочкой
 - ◆ Graphics: Load, (разделитель), Center – отключен, Stretch – отключен
3. Создайте всплывающее меню с единственным пунктом Clear.

Пример формы представлен на рис. 3.9.

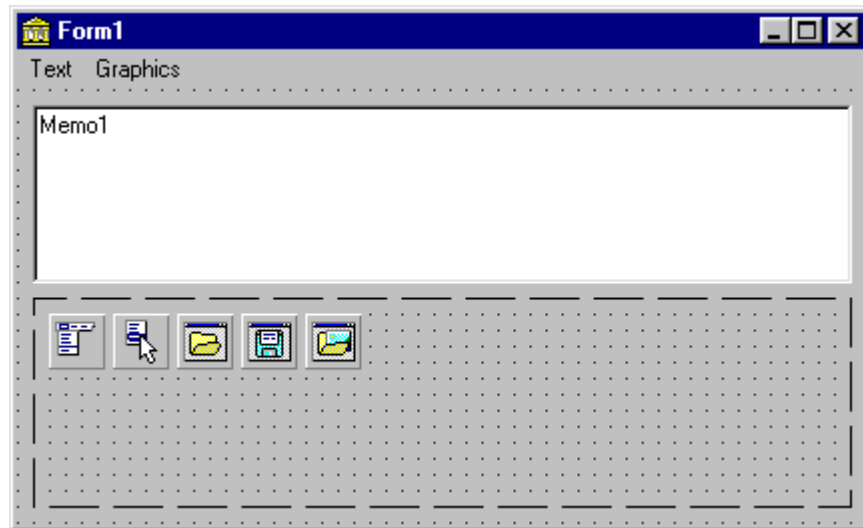


Рис. 3.9. Пример формы варианта 5

4. Коды методов:

```
procedure TForm1.Load1Click(Sender: TObject);
begin
  if OpenFileDialog1.Execute then
    Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
end;
```

```
procedure TForm1.Save1Click(Sender: TObject);
begin
  if SaveDialog1.Execute then
    Memo1.Lines.SaveToFile(SaveDialog1.FileName);
end;
```

```
procedure TForm1.PopupMenu1Popup(Sender: TObject);
begin
  Clear1.Visible := Memo1.Lines.Count <> 0;
end;
```

```
procedure TForm1.Clear1Click(Sender: TObject);
begin
  Memo1.Lines.Clear;
end;
```

```

procedure TForm1.Center1Click(Sender: TObject);
begin
  Center1.Checked := not Center1.Checked;
  Image1.Center := Center1.Checked;
end;

procedure TForm1.Load2Click(Sender: TObject);
begin
  if OpenPictureDialog1.Execute then
  begin
    Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName);
    Center1.Enabled := true;
    Stretch1.Enabled := true;
  end;
end;

procedure TForm1.Enabled1Click(Sender: TObject);
begin
  Enabled1.Checked := not Enabled1.Checked;
  Memo1.Enabled := Enabled1.Checked;
end;

```

5. Добавьте в главное меню пункт:

- ◆ Lines: Add, (разделитель) - невидимый

6. Добавьте методы:

```

procedure TForm1.Add1Click(Sender: TObject);
var
  I : TMenuItem;
begin
  I := TMenuItem.Create(Lines1);
  I.Caption := Memo1.SelText;
  I.OnClick := Put;
  Lines1.Add(I);
  N3.Visible := true;
end;

procedure TForm1.Put(Sender: TObject);
begin
  Memo1.Text := Memo1.Text + (Sender as TMenuItem).Caption;
end;

```

7. Добавьте в меню Text пункт "Alignment: Left", циклически изменяющий свойство Memo1.Alignment и своё свойство Caption для выравнивания по левому краю, по центру и по правому краю.

Вариант 6

1. На форме расположите три кнопки (вторая и третья - отключены), компоненты ProgressBar и Shape, пиктограммы диалога ColorDialog
2. Создайте меню:
 - ◆ Progress: More, Less - отключен
 - ◆ Buttons: First – с галочкой, Second, Third
 - ◆ Shaping: Color

Пример формы – рис. 3.10

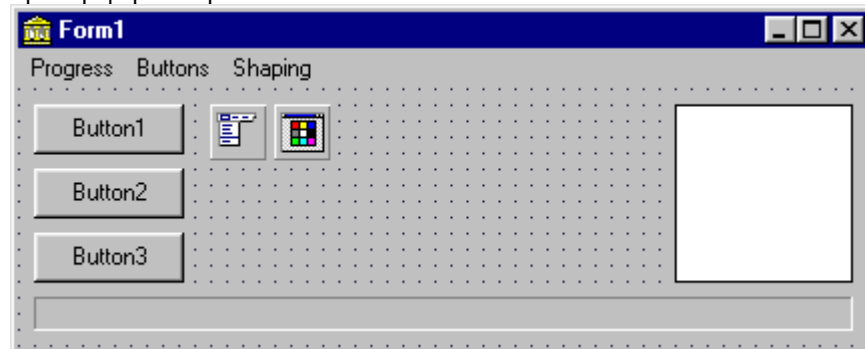


Рис. 3.10. Пример формы варианта 6

3. Коды методов:

```
procedure TForm1.More1Click(Sender: TObject);
begin
  ProgressBar1.Position := ProgressBar1.Position+10;
  Progress1.Caption := 'Progress: ' + IntToStr(ProgressBar1.Position) +
  '%';
  More1.Enabled := ProgressBar1.Position<>100;
  Less1.Enabled := ProgressBar1.Position<>0;
end;

procedure TForm1.First1Click(Sender: TObject);
begin
  First1.Checked := true;
```

```
Second1.Checked := false;  
Third1.Checked := false;  
Button1.Enabled := true;  
Button2.Enabled := false;  
Button3.Enabled := false;  
end;  
  
procedure TForm1.Color1Click(Sender: TObject);  
begin  
    if ColorDialog1.Execute then  
        Shape1.Brush.Color := ColorDialog1.Color;  
end;
```

4. Код метода TForm1.Less1Click аналогичен коду метода TForm1.More1Click
5. Коды методов TForm1.Second1Click и TForm1.Third1Click аналогичны коду метода TForm1.First1Click
6. Добавьте в меню Shaping пункт, циклически изменяющий свойство Shape1.Shape и отображающий эти изменения своим свойством Caption

6. Результаты работы

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, содержащий меню, файл формы и исходный код модуля.

ЛАБОРАТОРНАЯ РАБОТА №3

Классы и модули

1. Цель работы

Объектно – ориентированное программирование в среде Delphi и разработка приложения, создающего и обрабатывающего собственный класс.

2. Домашнее задание

Изучить раздел 3 конспекта лекций, описывающий объектно-ориентированное программирование в Delphi.

Ознакомиться с описанием и заданием на лабораторную работу.

3. Основные понятия и приемы

3.1. Классы и сокрытие информации

Класс может содержать сколько угодно данных и любое количество методов. Однако для соблюдения всех правил объектно-ориентированного подхода данные должны быть скрыты, или инкапсулированы, внутри использующего их класса. Использование метода для получения доступа к внутреннему представлению объекта уменьшает риск возникновения ошибочных ситуаций и позволяет автору класса модифицировать внутреннее представление в будущих версиях. В Object Pascal имеются две различные конструкции, подразумевающие инкапсуляцию, защиту и доступ к переменным: классы и модули. С классами связаны некоторые специальные ключевые слова – спецификаторы доступа:

- ◆ `private` – элементы интерфейса класса видны только в пределах модуля, в котором определен класс. Вне этого модуля `private`-элементы не видны и недоступны. Если в одном модуле создано несколько классов, то все они видят `private`-разделы друг друга.
- ◆ Раздел `public` не накладывает ограничений на область видимости перечисленных в нем полей, методов и свойств. Их можно вызывать в любом другом модуле программы.
- ◆ Раздел `published` не ограничивает область видимости, однако в нем перечислены свойства, которые должны быть доступны не только на этапе исполнения, но и на этапе конструирования программы. Раздел `published` используется при разработке компонентов. Без объявления раздел считается объявленным как `published`. Такой умалчиваемый раздел располагается в самом начале объявления класса любой формы и продолжается до первого объявленного раздела. В раздел `published` среда помещает описание вставляемых в форму компонентов. Сюда не нужно помещать собственные элементы или удалять элементы, вставленные средой.

- ♦ Раздел `protected` доступен только методам самого класса, а также любым потомкам, независимо от того, находятся ли они в том же модуле или нет.

Порядок следования разделов может быть любой, любой раздел может быть пустым.

3.2. Классы и модули

Приложения Delphi интенсивно используют модули. За каждой формой скрывается соответствующий ей модуль. Однако модули не обязаны иметь соответствующие формы.

Модуль содержит раздел `interface`, где объявлено все, что доступно для других модулей, и раздел `implementation` с реальным кодом. Наконец, модуль может иметь два необязательных раздела: `initialization` с некоторым кодом запуска, который выполняется при загрузке в память программы, использующей данный модуль, и `finalization`, который выполняется при завершении программы.

Предложение `uses` в начале раздела `interface` указывает к каким другим модулям мы должны получить доступ из раздела `interface` текущего модуля. Если же на другие модули необходимо сослаться из кода подпрограмм и методов, вы должны добавить новое предложение `uses` в начале раздела `implementation`.

В интерфейсе модуля можно объявить несколько различных элементов, в том числе процедуры, функции, глобальные переменные и типы данных. Также можно поместить в модуль класс. Delphi это делает автоматически при создании каждой формы. Чтобы создать новый не относящийся к форме модуль, выберите команду `File/New` и отметьте на странице `New` появившегося окна `Object Repository` элемент `Unit`.

3.3. Модули и область видимости

Область видимости идентификатора (переменной, процедуры, функции или типа данных) определяет ту часть кода, в которой доступен этот идентификатор. Основное правило состоит в том, что идентификатор является значимым только внутри его области видимости.

Если вы объявили идентификатор внутри блока определения процедуры, вы не сможете использовать данную переменную вне этой процедуры. Область видимости идентификатора охватывает всю процедуру, включая вложенные блоки.

Если вы объявили идентификатор в области реализации модуля, вы не можете применить его вне модуля, но можете использовать в любом блоке и процедуре, определенных внутри модуля. Если вы объявили идентификатор в интерфейсной части модуля, его область видимости распространяется на любой другой модуль, где объявлен идентификатор. Любой идентификатор, объявленный в интерфейсе модуля, является глобальным;

все другие идентификаторы принято называть локальными.

3.4. Модули и программы

Приложение Delphi создается из файлов исходного текста двух разных видов. Это один или несколько модулей и один файл программы.

Модули можно считать вторичными файлами, к которым обращается основная часть приложения – программа. На практике файл программы обычно является автоматически сгенерированным файлом с ограниченной ролью. Он нужен только для запуска программы, которая выполняет основную форму. Код файла программы, или файла проекта Delphi (DPR), можно отредактировать вручную или с помощью Project Manager и некоторых опций проекта.

3.5. Информация о типе на этапе выполнения

Правило языка Object Pascal о совместимости типов для классов-потомков позволяет вам использовать класс-потомок там, где ожидается класс - предок, обратное невозможно. Теперь предположим, что класс Dog содержит функцию Eat, которая отсутствует в классе Animal.

Если переменная MyAnimal ссылается на объект типа Dog, вызов этой функции должен быть разрешен. Но если вы попытаетесь вызвать эту функцию, а переменная ссылается на объект другого класса, возникнет ошибка. Поскольку компилятор не способен определить, будет ли значение правильным на этапе выполнения, то, делая явное приведение типов, мы рискуем вызвать опасную ошибку этапа выполнения программы.

Для решения данной проблемы можно воспользоваться некоторыми подходами, основанными на системе RTTI. Каждый объект знает свой тип и своего предка и может получить эту информацию с помощью операции is. Параметрами операции is являются объект и тип:

```
if MyAnimal is Dog then ...
```

Выражение is становится истинным, только если в настоящее время объект MyAnimal имеет тип Dog или тип потомка от Dog. Другими словами, это выражение приобретает значение True, если вы можете без риска присвоить объект (MyAnimal) переменной заданного типа данных (Dog). Такое прямое приведение можно выполнить так:

```
if MyAnimal is Dog then  
  MyDog := Dog (MyAnimal) ;
```

То же действие можно выполнить напрямую с помощью другой операции RTTI – as. Мы можем написать так:

```
MyDog := MyAnimal as Dog ;  
Text := MyDog. Eat ;
```

Если мы хотим вызвать функцию Eat, можно использовать и другую форму записи:

```
(MyAnimal as Dog) . Eat ;
```

Результатом выражения будет объект с типом данных класса Dog, поэтому вы можете применить к нему любой метод этого класса.

Приведение с операцией as отличается от традиционного приведения тем, что в случае несовместимости типа объекта с типом, к которому вы пытаетесь его привести, порождается исключение EInvalidCast.

Чтобы избежать этого исключения, используйте операцию is и в случае успеха делайте простое приведение:

```
if MyAnimal is Dog then  
  (Dog (MyAnimal)) . Eat ;
```

4. Порядок выполнения работы

1. В среде программирования Delphi создайте новый проект, выбрав пункт меню File/New Application.
2. Сохраните этот проект в папке "C:\Ваша_группа\Ваша_фамилия Lab4". (Unit1.pas под новым именем Main4.pas, а Project1.dpr под новым именем Lab4.dpr).
3. Открыть модуль, не связанный с формой (File/New и отметьте на странице New появившегося окна Object Repository элемент Unit), и поместить в него три класса:
 - ◆ Класс Animal, который содержит в разделе public объявление конструктора Create и объявление метода-функции: Voice – звук, издаваемый животным. Тип результата возвращаемого функцией, – string. Метод Voice объявить виртуальным и абстрактным.

```
public  
  constructor Create;  
  function GetKind: string;  
  function Voice: string; virtual; abstract;
```

В разделе private класса определить переменную Kind: string.

- ◆ Класс Dog объявить потомком класса Animal:
TDog = class (TAnimal)
 В разделе public этого класса объявить конструктор и методы Voice и Eat. Метод Eat типа string объявить виртуальным (пища животного).

```

public
constructor Create;
function Voice: string; override;
function Eat: string; virtual;

```

- ◆ Класс Cat объявить потомком класса Animal.

```
TCat = class (TAnimal)
```

Раздел public класса содержит те же определения, что и соответствующий раздел класса Dog.

```

public
constructor Create;
function Voice: string; override;
function Eat: string; virtual;

```

В реализациях конструктора каждого класса переменной Kind присваивается имя соответствующего животного, например, для класса Animal:

```
Kind := 'An Animal'.
```

В реализациях методов Voice возвращается звук, издаваемый животным, например:

```
Voice := 'Mieow'.
```

В реализациях методов Eat возвращается название пищи, которой питается соответствующее животное:

```
Eat:= 'A bone, please!';
```

4. Задать имя модуля и имя проекта, в который этот модуль будет включен.
7. Добавить в проект форму, которой присвоить имя Animals, также задать имя модулю, связанному с формой.
8. В форме расположить три кнопки опций (компонент RadioButton) с названиями Animal, Dog, Cat ; кнопкой команды (компонент Button) с названием Kind и две крупные надписи (компонент Label)(рис. 4.1). Нажатие одной из кнопок опций будет соответствовать выбору животного. При нажатии кнопки команды надписи должны отобразить звук, издаваемый животным, и его пищу.



Рис. 4.1. Форма проекта

9. Определите в классе формы private-переменную MyAnimal.

```
private
MyAnimal: TAnimal;
```

Запишите код для обработчика события OnCreate формы, где создается объект типа Dog, на который ссылается переменная MyAnimal.

```
begin
MyAnimal := TDog.Create;
end;
```
10. В обработчиках события OnClick каждой кнопки опций записать код, который удаляет текущий объект и создает новый.

```
begin
MyAnimal.Free;
MyAnimal := TAnimal.Create;
end;
```
11. В обработчике события OnClick кнопки команды записать код, который будет помещать в надписи звук, издаваемый животным.

```
begin
LabelVoice.Caption := MyAnimal.Voice;
end;
```



Рис. 4.2. Звук для Cat

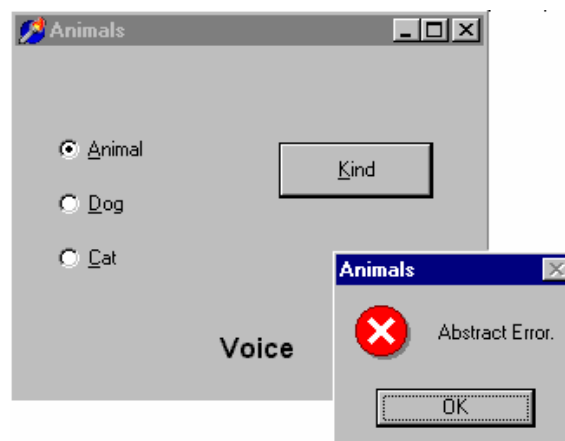


Рис. 4.3. Сообщение об ошибке

12. Если вы все сделали правильно, при запуске приложения надписи будут отображать пищу и звук для Dog и Cat (рис. 4.2) и приложение завершит работу по ошибке при выборе Animal (рис. 4.3).
13. Уберите ключевое слово `abstract` в объявлении метода `Voice`. Запустите приложение снова. Посмотрите, что изменилось в работе приложения. Объясните различия.
14. Попробуйте использовать метод `Eat` без приведения типов (без `is`).
15. Доработайте проект согласно варианту индивидуального задания.

5. Варианты индивидуальных заданий

Вариант 1

Разработайте два класса потомка от `Animal`, которые будут отображать особенности двух пород собак. Разработайте методы для этих классов, позволяющие получить некоторые характеристики породы (рост, длина шерсти, длина ушей и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

Примерная форма проекта представлена на рис. 4.4.

После запуска программы представляется возможность выбора между значениями: `Animal`, `Dog`, `Cat`. Выбор осуществляется при помощи кнопки `SELECT`. Далее предоставляется дополнительная возможность для выбора породы собаки.

При выделении знаком:



интересующей породы можно получить дополнительную информацию.

Вариант 2

Разработайте два класса потомка от `Animal`, которые будут отображать особенности двух новых животных `Wolf` (волк) и `Jascal` (шакал). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих видов животных (рост по холке, длина клыков, вес и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

Вариант 3

Разработайте два класса потомка от `Animal`, которые будут отображать особенности двух новых животных `Fish` (рыба) и `Bird` (птица). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих типов животных (среда обитания, покров тела, издаваемый звук и т.д.). Дополните форму компонентами, позволяющими увидеть все характеристики разработанных классов.

Вариант 4

Разработайте два класса потомка от `Animal`, которые будут отображать особенности двух видов насекомых `Gnat` (комар) и `Fly` (муха). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих насекомых (скорость передвижения, окраска, используемая пища и т.д.). Дополните форму компонентами, позволяющими увидеть все эти характеристики разработанных классов.

Вариант 5

Разработайте два класса потомка от `Animal`, которые будут отображать особенности двух видов млекопитающих `Man` (человек) и `Monkey` (обезьяна). Разработайте методы для этих классов, позволяющие получить некоторые характеристики этих существ (способ общения, покров, рост и т.д.). До-

полните форму компонентами, позволяющими увидеть все эти характеристики разработанных классов.

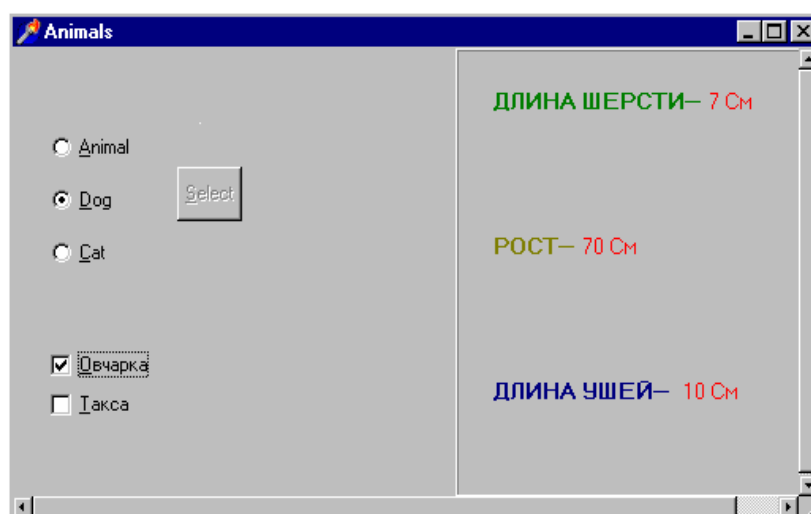


Рис. 4.4. Форма характеристик пород собак

6. Результаты работы

В результате выполнения лабораторной работы студент должен продемонстрировать преподавателю готовый проект, файлы форм и исходный код модулей.